VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe
Individual Professional Practise in the Company

2010 Kamil Grebeníček

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2010 Kamil Grebeníček

Abstrakt

Bakalářská práce se zabývá úkoly zadanými a zpracovanými v průběhu individuální odborné praxe, kterou jsem vykonával ve společnosti KVADOS, a.s., se sídlem Novoveská 1139/22, 709 00 Ostrava – Mariánské Hory. Práce je rozdělena na dvě hlavní části. První část popisuje zadané úkoly a jejich základní teorii. Ve druhé části se věnuji postupům, které jsem při řešení úkolů zvolil. K práci jsem používal balík nástrojů pro řízení životního cyklu softwarových aplikací Microsoft Visual Studio 2010. Konkrétně se zabývám nástroji Microsoft Visual Studio 2010 Ultimate Web Testing, Microsoft Test Manager a systémem pro automatické sestavení aplikací Team Foundation Build platformy Microsoft Visual Studio Team Foundation Server 2010. V závěrečné části práce vysvětluji, jaké znalosti jsem měl před započetím praxe a jaké jsem nabyl během ní.

Abstract

The Bachelor thesis describes assigned tasks the author had to solve on Individual Professional Practise in KVADOS, a.s. Company based Novoveská 1139/22, 709 00 Ostrava – Mariánské Hory. This thesis contains two main sections. First section describes assigned tasks and some basic theory. Second section explains procedures the author chooses for solving the tasks. During Individual Professional Practise Microsoft Visual Studio 2010 Application Lifecycle Management was used. In the concrete the author works with tools Microsoft Visual Studio 2010 Ultimate Web Testing, Microsoft Test Manager and build automation component of Microsoft Visual Studio Team Foundation Server 2010 platform Team Foundation Build. The final section of this work describes knowledge and skills the author already had and acquired during Individual Professional Practise.

Klíčová slova

.NET, C#, Microsoft Visual Studio, ASP.NET, testování webových aplikací, Web Testing, Web Performance Test, Microsoft Test Manager, MSBuild, Team Foundation Build, TFS

Keywords

.NET, C#, Microsoft Visual Studio, ASP.NET, web application testing, Web Testing, Web Performance Test, Microsoft Test Manager, MSBuild, Team Foundation Build, TFS

Děkuji svému konzultantovi, Radku Garzinovi, že mi umožnil získat cenné zkušenosti během výkonu stáže.

Seznam použitých zkratek

ADO.NET	ActiveX Data Objects .NET
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASP.NET	Active Server Pages .NET
CSV	Comma-separated Values
DOM	Document Object Model
ERP	Enterprise Resource Planning
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IIS	Internet Information Service
IP	Internet Protocol
Ms-PL	Microsoft Public Licence
SQL	Structured Query Language
TFS	Team Foundation Server
UI	User Interface
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

Obsah

1 Úvod

Bakalářská práce se věnuje popisu úkolů a jejich řešení, které mi byly zadány v průběhu absolvování individuální odborné praxe probíhající poslední dva semestry bakalářského studia. Zadané úkoly se dají rozdělit do tří oblastí a každé této oblasti budu věnovat adekvátní část práce podle celkové časové náročnosti řešeného úkolu. Nejdéle se pozastavíme u tématu týkajícího se funkčního testování webových aplikací na platformě .NET pomocí komplexní sady pro řízení životního cyklu softwarových aplikací Microsoft Visual Studio 2010 Ultimate. Další oblast se zabývá možnostmi, jež nabízí specializovaná sada nástrojů pro odborníky na zajištění kvality usnadňující plánování a spouštění manuálních i automatizovaných testů Microsoft Visual Studio Test Professional 2010 a v poslední oblasti prozkoumáme některé možnosti sestavování aplikací s pomocí platformy pro spolupráci v rámci řešení pro správu životního cyklu aplikace Microsoft Visual Studio Team Foundation Server 2010.

Práce je rozdělena do několika částí, kde první část popisuje odborné zaměření firmy, u které jsem vykonával odbornou praxi a mé pracovní zařazení. Krátce uvedu profil společnosti, její historii i současné postavení na trhu a stručný popis produktového portfolia. V druhé části práce se věnuji konkrétním zadaným úkolům. Popíši tedy problém, objasním teorii týkající se zadaných úkolů, kterou bylo nutné nastudovat, zvládnout i pochopit a u každého úkolu také uvedu přibližné množství stráveného času. Ve třetí části práce se již zaměřím na řešení zadaných úkolů, popíši veškeré záležitosti a postřehy, se kterými jsem se při jejich řešení setkal a předvedu zvolený postup řešení úkolů, který také vysvětlím, popř. uvedu zdroje, jež mi s jeho řešením pomohly. Konečně v poslední části práce uvedu teoretické i praktické znalosti a dovednosti získané v průběhu studia, které jsem během odborné praxe uplatnil a také ty, jež mi při výkonu odborné praxe chyběly. Pokusím se o celkové zhodnocení průběhu odborné praxe a představení dosažených výsledků.

2 Profil společnosti a pracovní zařazení

Společnost KVADOS, a.s., je významným systémovým integrátorem a výrobcem vlastních informačních systémů. Klíčové produktové jsou určeny společnostem působícím v obchodním, distribučním, servisním nebo logistickém sektoru. KVADOS, a.s., aktivně působí na většině evropských trhů. Využívá k tomu vlastních obchodních zastoupení i rozsáhlé sítě obchodních partnerů.[1]

Historie KVADOS, a.s., sahá do roku 1992. Od svého založení se společnost vypracovala z pozice malého regionálního softwarového výrobce k významnému systémovému integrátorovi s evropským působením a producentovi softwarových řešení, která se s úspěchem používají v celé Evropě. Na úplném počátku se společnost orientovala na distribuci produktů třetích stran. Již záhy se však zaměřila na vývoj a implementaci vlastních řešení a tato cesta se ukázala jako správná. Hlavní orientací dodnes zůstal vývoj komplexních informačních systémů pro obchodní a obchodně-výrobní společnosti.[1]

Skupina KVADOS je výrobcem vlastních produktových řad:[1]

- **VENTUS** – komplexní řešení pro obchodní a obchodně-výrobní společnosti (ERP systém)
- **myAVIS™** – mobilní informační systém pro uživatele, kteří pracují v odvětví obchodních, servisních nebo distribučních služeb a působí především mimo kancelář
- **myCASH™** – pokladní systém přinášející nejmodernější řešení pro jednotlivé prodejny i celé maloobchodní sítě
- **myWORK™** – informační systém pro analýzu procesů a činností zaměstnanců ve společnostech, který zvyšuje efektivitu pracovních týmů a konkurenceschopnost

Při výkonu odborné praxe ve společnosti KVADOS, a.s. jsem byl zařazen do oddělení vývoje webových aplikací pod vedoucím Radkem Garzinou, který mi zadával úkoly, poskytoval konzultace a jemuž jsem také prezentoval dosažené výsledky.

3 Zadané úkoly

Zadané úkoly se netýkaly práce na žádném konkrétním projektu společnosti, ale jednalo se spíše o prozkoumání určité oblasti, nalezení vhodné technologie či nástroje a jejich odzkoušení na reálném příkladu, čímž se prověřila jejich vhodnost z hlediska požadavků. V počáteční fázi řešení úkolů se tedy jednalo výhradně o hledání a studium vhodných materiálů a celkové pochopení a zvládnutí problematiky. V dalších fázích řešení úkolů se již jednalo o postupné prověřování specifických vlastností nebo funkcí zvoleného nástroje z pohledu nasazení na reálných projektech společnosti.

Následující podkapitoly jsou rozděleny podle zadaných úkolů. V každé z nich si podrobněji probereme problematiku a teorii řešeného problému.

3.1 Systém pro automatické testování webových aplikací

Primárním úkolem odborné stáže bylo vytvoření jakéhosi systému pro automatické testování webových aplikací. Testování webové aplikace by mělo probíhat na vrstvě protokolu HTTP, kde se odfiltrují a zaznamenají jednotlivé chybové stavy, které nastanou při požadavcích odeslaných webovému serveru aplikace. Tento systém by měl být schopen zaznamenat, uchovat a znovu vykonat sérii požadavků odeslaných webové aplikaci. Tyto série požadavků odpovídají jednotlivým testovacím scénářům a jejich uchováním, možným znovupoužitím a plánovaným nebo ručním vykonáním by se docílilo automatizovaného provádění testů. Mělo by se tedy jednat o jakousi proxy zachytávající komunikaci mezi klientem a webovým serverem. Série požadavků klienta se zaznamenají pro znovupoužití jako jednotlivé testovací scénáře a odpovědi webového serveru na tyto požadavky se uloží do databáze jako výsledky jednotlivých testovacích scénářů.

Moje práce na úkolu začala studiem informací týkajících se funkčního testování webových aplikací. Po nabytí základního povědomí o této problematice jsem začal implementovat testovací webovou aplikaci, pomocí které bych si ověřil nabyté poznatky na praktickém příkladě. Podle návrhů svého konzultanta jsem vytvořil jednoduchou webovou aplikaci ASP.NET, která pomocí třídy HttpListener rozhraní .NET Framework odposlouchávala a serializovala požadavky klientů do databáze. To odpovídalo prvotní fázi ukládání sérií požadavků jako testovacích scénářů pro jejich opětovné vyvolání, což bylo zmiňováno dříve.

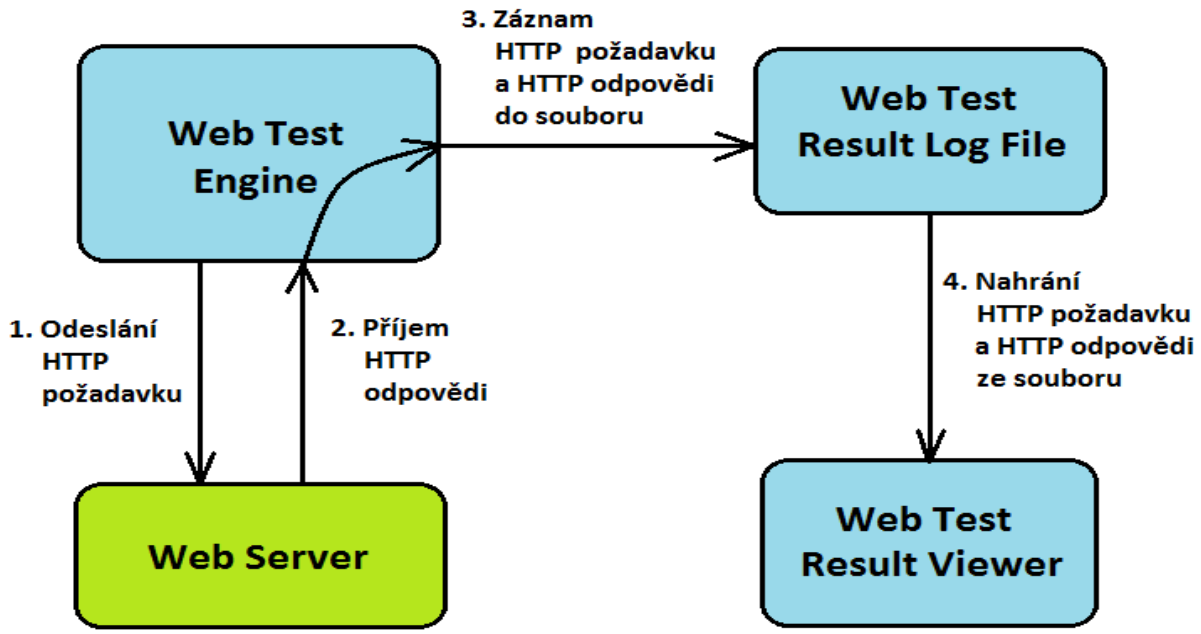
Při hledání informací týkajících se tématu jsem samozřejmě nemohl minout některé komerční produkty různých společností zaměřené na široké oblasti testování webových aplikací. Nákup a nasazení takového produktu by si jistě vyžádal vynaložení nemalých prostředků, což nepřipadalo v úvahu. Proto objevení vhodného nástroje přímo od společnosti Microsoft, který je navíc integrován v již nasazené platformě pro spolupráci Visual Studio TFS, získalo velkou pozornost. Řešení úkolu se

tedy v podstatě přeorientovalo ze snahy o vytvoření vlastního systému na snahu o co nejvhodnější využití již existujícího řešení, kterému se budu věnovat po zbytek této podkapitoly.

3.1.1 Microsoft Visual Studio 2010 Ultimate Web Testing

Společnost Microsoft ve své sadě Visual Studio 2010 Ultimate nabízí vestavěné nástroje určené k automatickému funkčnímu testování webových aplikací a odpovídající typ testu, který lze přidat do testovacího projektu sady Visual Studio. Tento typ testu byl poprvé uveden v sadě Visual Studio 2005 pod názvem Web Test a v nejnovější verzi sady byl jeho název změněn na Web Performance Test. Běžně se však dále tento typ testu označuje původním názvem a stejně tak tomu bude i v této práci.

Ačkoliv lze Web Testy s výhodou použít pro funkční testování, není to jejich primárně zamýšlený účel. Tím je jejich použití jako skriptů simulujících činnost uživatele při zátěžových testech. Zátěžové testy se používají pro generování požadavků virtuálních uživatelů oproti webovému serveru, na kterém aplikace běží. Poté se měří odezva serveru na jednotlivé požadavky a zaznamenávají se výskyty chybových odpovědí vrácených serverem. Při zátěžových testech je nutné generovat co nejvyšší zátěž s co nejnižší režií na co nejmenším množství hardwaru. Z tohoto důvodu Web Testy pracují na vrstvě protokolu HTTP a při svém vykonávání nespouštějí webový prohlížeč. Následující obrázek zobrazuje schéma vykonávání Web Testu, kde si lze všimnout, že v žádné fázi běhu testu se webový prohlížeč nespouští.[2]



Obr. 3-1: Schéma vykonávání Web Testu[2]

Jak lze vidět z předchozího obrázku, hlavní součástí vykonávající test je Web Test Engine, který odesílá požadavky ze série uložené v souboru testu na webový server, zaznamenává odpovědi webového serveru, měří časové údaje a tyto data uchovává ve Web Test Result Log souboru. Prohlížeč výsledků Web Test Result Viewer poté nahrává tyto data z Web Test Result Log souboru a zobrazuje je uživateli.

Protože se při vykonávání testu nespouští webový prohlížeč, je nutné si uvědomit některé z toho plynoucí důsledky. Jak již bylo několikrát uvedeno, Web Testy pracují na vrstvě protokolu HTTP v tom smyslu, že na webový server se odesílají zaznamenané HTTP požadavky a uchovávají se přijaté HTTP odpovědi. To znamená, že se žádné požadavky klientského kódu (JavaScript, popř. Ajax) nebo zásuvného modulu prohlížeče (např. Silverlight nebo Flash) během nahrávání testu nezaznamenají a tedy při vykonávání testu ani neodesílají. To je při práci s Web Testy velice důležité mít na paměti. Ve většině případů lze tyto problémy vyřešit manuální přidáním chybějících požadavků do Web Testu.[2]

V následujících oddílech probereme základní informace týkající se vytvoření Web Testu a možností jeho úprav, včetně přidání rozšiřující logiky.

3.1.1.1 Vytvoření Web Testu

Před vytvořením Web Testu je nutné mít vytvořen testovací projekt sady Visual Studio a do tohoto testovacího projektu lze poté přidat novou položku typu Web Performance Test, popř. Web Test ve starších verzích sady Visual Studio. Tímto se do testovacího projektu přidá nový soubor s příponou .webtest, odpovídající nově vytvořenému testu a automaticky se spustí webový prohlížeč Microsoft Internet Explorer. Spuštěný prohlížeč obsahuje podokno s názvem Web Test Recorder zaznamenávající požadavky uživatele a každý odeslaný požadavek zobrazí ve svém podokně. Mezi jednotlivé požadavky lze umístit komentáře napomáhající orientaci ve Web Testu, odstranit všechny již zaznamenané požadavky, popř. kliknutím na tlačítko Pause pozastavit záznam testu. Kliknutím na tlačítko Stop se záznam ukončí, okno prohlížeče se zavře a všechny zaznamenané požadavky se zobrazí v okně sady Visual Studio, které se nazývá Web Test Editor. Tento editor zobrazuje seznam požadavků, kde každý z nich se skládá z URL adresy zdroje webového serveru, seznamu parametrů dotazu a v případě, že se jedná o HTTP požadavek odeslaný metodou POST, tak i seznam parametrů formuláře. Prostřednictvím tohoto editoru lze s testem dále pracovat, upravovat jej a přidávat další možnosti, které si nyní představíme.

3.1.1.2 Modifikace Web Testu

Web Test Editor umožňuje Web Testu přidat určitou logiku i další funkčnost, pomocí které lze test upravit podle požadavků a tak využít jeho možností, čili nejedná se pouze o jednoduché zaznamenání série požadavků, jejich možné znovu vyvolání a uchování vrácených odpovědí. Díky své

rozšiřitelnosti lze s Web Testy dosáhnout mnohem více a tak upravit chování a průběh testu podle potřeb testované aplikace. Možností úprav je mnoho a těmi nejdůležitějšími se budeme zabývat ve zbytku tohoto oddílu.

Požadavky zaznamenané při vytvoření Web Testu obsahují všechny parametry dotazu a parametry formuláře použité při odeslání požadavku na webový server. Ve Web Test Editoru však lze požadavkům přidat tyto parametry vlastní. Dále lze požadavku přidat vlastní hlavičky, závislé požadavky a parametry pro nahrání souboru na server.

Jednotlivým zaznamenaným požadavkům lze také přidat pravidla dvojího typu a to pravidla typu Validation Rule a pravidla typu Extraction Rule. Pomocí pravidel typu Validation Rule lze validovat obsah odpovědi vrácené serverem. Pokud tedy požadavku přidáme pravidlo typu Validation Rule validujeme obsah jeho vlastní odpovědi. Splňuje-li odpověď podmínky definované pravidlem typu Validation Rule, bude jeho výsledkem úspěch, v opačném případě neúspěch. Neúspěšný výsledek pravidla typu Validation Rule způsobí, že požadavek jako jednotlivý krok testu selže, ve výsledku testu bude označen jako chyba a test skončí jako neúspěšný. Pravidlem typu Validation Rule lze validovat např. přítomnost konkrétního HTML elementu v odpovědi, existenci požadovaného textu v odpovědi a podobně.

Pravidla typu Extraction Rule jsou naproti tomu určena k vyextrahování určité části nebo hodnoty z odpovědi na požadavek, jež má pravidlo typu Extraction Rule aplikováno. Vyextrahovaná hodnota je poté uložena jako parametr do kontextu Web Testu, což je kolekce dvojic název/hodnota určená pro ukládání parametrů kontextu získaných nejčastěji pomocí pravidla typu Extraction Rule. Názvy parametrů kontextu lze poté využít namísto hodnot ostatních položek Web Testu, kdy se za běhu testu dosadí jejich skutečné hodnoty. Parametr kontextu získaný pomocí pravidla typu Extraction Rule z určitého požadavku je ostatním položkám Web Testu dostupný od následujícího požadavku v pořadí. Parametry kontextu lze taktéž vytvořit manuálně ve Web Test Editoru, jenž jsou na rozdíl od parametrů kontextu získaných pravidly typu Extraction Rule za běhu ihned dostupné všem položkám testu. Pravidlem typu Extraction Rule lze do kontextu Web Testu vyextrahovat např. vnitřní text nebo hodnotu atributu HTML elementu obsahu odpovědi a tuto získanou hodnotu následně použít např. jako hodnotu některé vlastnosti pravidla typu Validation Rule. Syntaxe pro použití parametru kontextu jako hodnoty položky nebo vlastnosti testu je následující:

{{názevParametruKontextu}}

Název parametru kontextu jednoduše vložíme do dvojitých složených závorek.

Existence kontextu Web Testu také umožňuje parametrizovat název (tj. doménové jméno nebo IP adresu) webového serveru, díky čemuž lze Web Testy zaznamenat oproti aplikaci běžící na vývojovém webovém serveru a později pouhou změnou hodnoty parametru změnit část názvu počítače URL adresy všech požadavků v testu a tak již jednou nahrané testy využít pro testování aplikace nasazené na jiný, např. testovací nebo produkční webový server.

Web Testy mohou také obsahovat data binding, pomocí kterého lze zajistit vstup dat z následujících datových zdrojů:[3]

- Databáze systému Microsoft SQL Server
- Databáze aplikace Microsoft Access
- Sešit aplikace Microsoft Excel
- Dokument XML
- Soubor ve formátu CSV

Po kliknutí na tlačítko Add Data Source ve Web Test Editoru se zobrazí okno průvodce umožňující vybrat typ datového zdroje a provést jeho následnou konfiguraci. Takto vytvořený datový zdroj je poté přidán do kontextu Web Testu jako parametr kontextu a pomocí něj lze přistupovat k datům datového zdroje. Po spuštění testu je název parametru kontextu odpovídající sloupci tabulky datového zdroje nahrazen svou hodnotou. Syntaxe pro použití datového zdroje jako parametru kontextu je následující:

{{názevDatovéhoZdroje.názevTabulky.názevSloupce}}

Datovému zdroji lze nakonfigurovat následující metody přístupu k datům, což je způsob jakým test načítá data z datového zdroje:[3]

- **Statická** – při spuštění testu se provede jediná iterace a použije se pouze první záznam datového zdroje.
- **Sekvenční** – při spuštění testu se začne načtením prvního záznamu, při každé další iteraci je nahrán následující záznam datového zdroje. Po nahrání posledního záznamu se opět načte první záznam a proces se opakuje, dokud neskončí zátěžový test, se kterým je Web Test asociován.
- **Náhodná** – při spuštění testu se pro každou iteraci náhodně vybere záznam z datového zdroje. Proces se opakuje, dokud neskončí zátěžový test, se kterým je Web Test asociován.
- **Unikátní** – při spuštění testu se začne načtením prvního záznamu a provede se tolik iterací, kolik je záznamů v datovém zdroji. Při každé další iteraci je nahrán následující záznam datového zdroje a po nahrání posledního záznamu se test ukončí.

Další modifikace nahraného Web Testu lze provést přidáním větvení nebo iterací. Pomocí větvení lze za běhu testu na základě podmínky definované při modifikaci testu rozhodnout, zda se jeho určitá část vykoná nebo nevykoná. Pomocí iterace lze zase za běhu opakovat určitou část testu, dokud je splněna podmínka definovaná při modifikaci testu. Funkcionalita větvení a iterací byla do Web Testu přidána až v posledním vydání sady Visual Studio. V předchozích verzích sady toto bylo možné pouze pomocí Coded Web Testu, který si krátce představíme v následujícím oddíle.

3.1.1.3 Coded Web Test

Z Web Test Editoru lze stisknutím tlačítka Generate Code vygenerovat tzv. Coded Web Test, což je ekvivalent aktuálně otevřeného Web Testu reprezentovaný jako třída v jazyce C# nebo Visual Basic. Veškeré možnosti vytvoření a modifikace Web Testu v uživatelském rozhraní Web Test Editoru nejsou ničím jiným než grafickou nadstavbou nad rozhraním API Web Testu, jenž je veřejně dostupné a nic kromě potřebných znalostí nám tedy nebrání napsat si vlastní Coded Web Test.

Coded Web Test je třída rozhraní .NET Framework odvozená z abstraktní třídy WebTest, která se nachází ve jmenném prostoru Microsoft.VisualStudio.TestTools.WebTesting assembly Microsoft.VisualStudio.QualityTools.WebTestFramework.dll. Třída Coded Web Testu musí přepsat abstraktní metodu GetRequestEnumerator své základní třídy vracející generické rozhraní IEnumerator<WebTestRequest>. Tato metoda při každém volání vytvoří instanci třídy WebTestRequest, naplní ji požadovanými daty a poté pomocí klíčového slova yield vrátí referenci na tento objekt volajícímu.

Předpokládá se, že Web Testy nebudou vytvářet pouze vývojáři, ale např. i pracovníci zajišťující kvalitu vyvíjené aplikace, pro které je tedy spíše určeno „klikací“ prostředí Web Test Editoru, kde při vytváření nebo modifikaci testu není nutné napsat jediný řádek kódu. Naproti tomu vývojáři mohou využít znalostí v oblasti programování a tedy takřka neomezených možností modifikace Coded Web Testu.

3.1.1.4 Spuštění Web Testu

Po vytvoření a úpravách testu je čas na jeho první spuštění a odzkoušení. To lze provést kliknutím na tlačítko Run Test v okně Web Test Editoru, resp. kliknutím pravým tlačítkem myši v okně editoru třídy Coded Web Testu a vybráním Run Tests. Zobrazí se okno Web Test Result Viewer zobrazující průběh vykonávání testu rozdělené do dvou částí. Horní část okna zobrazuje seznam požadavků testu a informace o jejich výsledku, poté co jsou úspěšně nebo neúspěšně vykonány. Kliknutím na vykonaný požadavek testu se v dolní části okna Web Test Editoru zobrazí jeho detailní informace prostřednictvím pěti karet. Okno první karty obsahuje komponentu webový prohlížeč vykreslující HTML kód odpovědi. Tato komponenta má zakázáno spouštění klientského kódu. Druhá a třetí karta obsahuje textový výpis hlaviček a parametrů odeslaného požadavku, resp. hlaviček a těla přijaté

odpovědi. Čtvrtá karta zobrazuje tabulku názvů a hodnot všech aktuálních parametrů v kontextu Web Testu. A poslední pátá karta obsahuje podrobnosti o výsledku všech pravidel aplikovaných na daný požadavek.

3.1.1.5 Rozšiřitelnost Web Testu

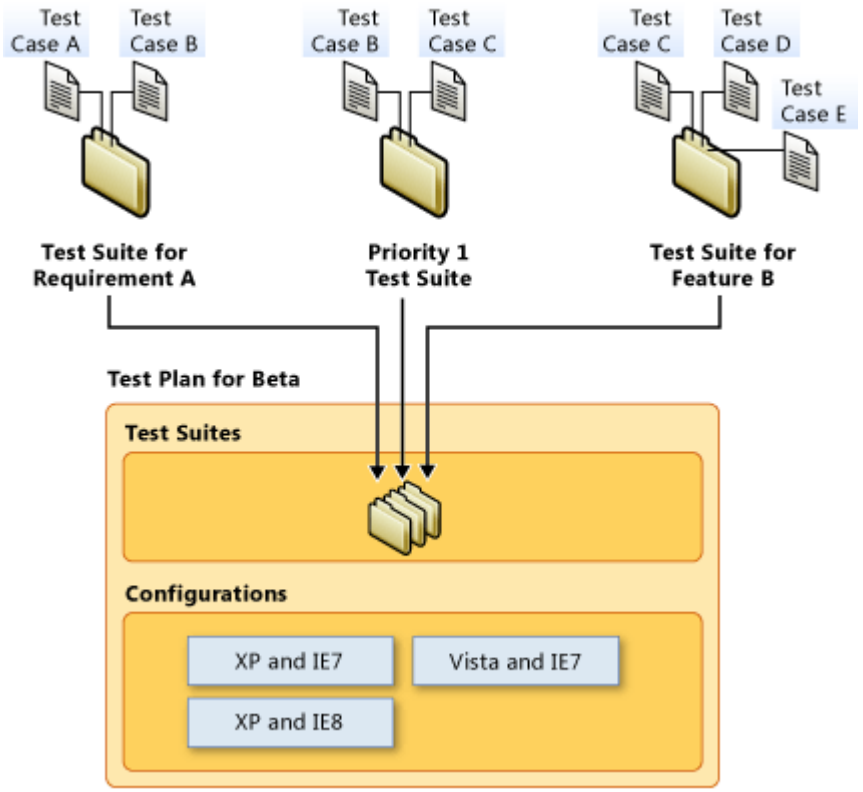
V tomto oddíle si představíme možnosti rozšíření Web Testu, které mu umožňují přidat vlastní funkcionalitu využitelnou v prostředí Web Test Editoru a tak se nespoléhat pouze na možnosti nabízené výrobcem. Rozhraní API Web Testu poskytuje široké možnosti rozšíření, z nichž všechny vyžadují znalost programování v jazyce C# nebo Visual Basic. Některé základní možnosti si krátce představíme v následujícím seznamu.

- **Custom Validation Rule** – vlastní pravidlo typu Validation Rule lze vytvořit pomocí veřejné třídy odvozené z abstraktní třídy ValidationRule a přepsáním její metody Validate, ve které nastavíme vlastnost IsValid objektu argumentu události na hodnotu true, pokud validace prošla, resp. na hodnotu false, pokud validace neprošla.
- **Custom Extraction Rule** – vlastní pravidlo typu Extraction Rule lze vytvořit pomocí veřejné třídy odvozené z abstraktní třídy ExtractionRule a přepsáním její metody Extract, ve které vyextrahujeme požadovanou hodnotu a uložíme ji do kontextu Web Testu.
- **Custom Conditional Rule** – vlastní pravidlo pro větvení nebo iteraci lze vytvořit pomocí veřejné třídy odvozené z abstraktní třídy ConditionalRule a přepsáním abstraktní metody CheckCondition, která určuje hodnotu podmínky větvení nebo iterace.
- **Web Test Request Plug-in** – zásuvný modul umožňující vykonat vlastní kód před nebo po každém požadavku Web Testu lze vytvořit pomocí veřejné třídy odvozené z abstraktní třídy WebTestRequestPlugin a přepsáním její metody PreRequest nebo metody PostRequest.
- **Web Test Plug-in** – zásuvný modul umožňující vykonat vlastní kód před začátkem nebo po skončení Web Testu lze vytvořit pomocí veřejné třídy odvozené z abstraktní třídy WebTestPlugin a přepsáním její metody PreWebTest nebo metody PostWebTest.
- **Web Test Recorder Plug-in** – zásuvný modul umožňující vykonat vlastní kód po ukončení nahrávání testu Web Test Recorderem lze vytvořit pomocí veřejné třídy odvozené z abstraktní třídy WebTestRecorderPlugin a přepsáním její metody PostWebTestRecording.

Všechny vlastní třídy je nutné umístit přímo do testovacího projektu nebo vytvořit knihovnu tříd a do testovacího projektu přidat její referenci. Tato vlastní funkcionalita je poté dostupná prostřednictvím uživatelského rozhraní Web Test Editoru. Ukázku vlastního pravidla si názorně na kódu představíme v kapitole Řešení úkolů.

3.2 Microsoft Test Manager

Poslední vydání balíku nástrojů pro řízení životního cyklu softwarových aplikací Microsoft Visual Studio 2010 přináší nový nástroj Microsoft Test Manager pro testery umožňující vytvářet a spravovat testovací plány, testovací sady, testovací scénáře, testovací konfigurace, testovací nastavení a testovací prostředí. Konkrétně je nástroj Microsoft Test Manager součástí produktů Microsoft Visual Studio 2010 Ultimate a Microsoft Visual Studio Test Professional 2010, které také obsahují nástroje určené k instalaci testovacích řadičů a testovacích agentů. Následující obrázek znázorňuje součásti testovacího plánu nástroje Microsoft Test Manager:



Obr. 3-2: Součásti testovacího plánu[4]

Pomocí nástroje Microsoft Test Manager lze do teamového projektu platformy Visual Studio TFS přidávat jednotlivé testovací plány skládající se z testovacích sad a testovacích konfigurací. Testovací sada obsahuje jeden nebo více testovacích scénářů, kde každý scénář odpovídá jednomu manuálnímu nebo automatickému testu (např. Web Test je automatický test). Testovací konfigurace umožňují při vykonání testovacího plánu nasimulovat různé konfigurace verze operačního systému Microsoft Windows a webového prohlížeče Microsoft Internet Explorer.

Při běhu testu lze sbírat různá data a diagnostické informace, velmi užitečné při následném hledání a opravování chyby, zachycené neúspěšným testem, což lze definovat pomocí testovacích nastavení testovacího plánu. V testovacím nastavení vybíráme a konfigurujeme mimo jiné diagnostické datové adaptéry určující chování testovacího počítače při vykonávání testovacího plánu. Například lze nastavit diagnostické datové adaptéry simulující úzké hrdlo v síti, provádějící sběr informací o systému, na kterém se testuje, nebo zaznamenávající video s děním na obrazovce během provádění manuálního testu.[4]

Testovací řadič je počítač asociovaný s kolekcí týmových projektů platformy Visual Studio TFS a jedním nebo více testovacími agenty, jimž rozděluje úlohy týkající se vykonávání testovacích plánů. Testovací agent je počítač, který vykonává úlohy přidělené testovacím řadičem. Testovací řadiče, testovací agenty a další lze instalovat pomocí nástroje Microsoft Visual Studio Agents 2010, který je součástí všech edicí integrovaného vývojového prostředí Visual Studio 2010.

Pomocí nástroje Microsoft Test Manager lze vytvořit fyzické nebo virtuální testovací prostředí pro běh testovacích plánů. Fyzické prostředí se může skládat z fyzických nebo virtualizovaných počítačů, jenž mají nainstalován software testovacího agenta. Virtuální prostředí se může skládat pouze z virtualizovaných počítačů se softwarem testovacího agenta. Výhodou virtuálního prostředí je možnost vytvoření snapshotu aktuálního stavu testovacího agenta vykonávajícího test a později při hledání a opravování chyby neúspěšného testu vývojářem, obnovit zaznamenaný stav testovacího agenta. Díky této funkci má vývojář k dispozici živý obraz počítače při vzniku chyby a např. si může k testované aplikaci připojit debugger a efektivně nalézt chybu v okamžiku jejího vzniku.

Nové nástroj balíku Visual Studio 2010 pro testování poskytuje mnoho možností a funkcí pro pohodlné a efektivní testování aplikace, z nichž jsme probrali ty základní. Další možnosti a funkce nepředstavené v této podkapitole jsou nad rámec tohoto stručného popisu.

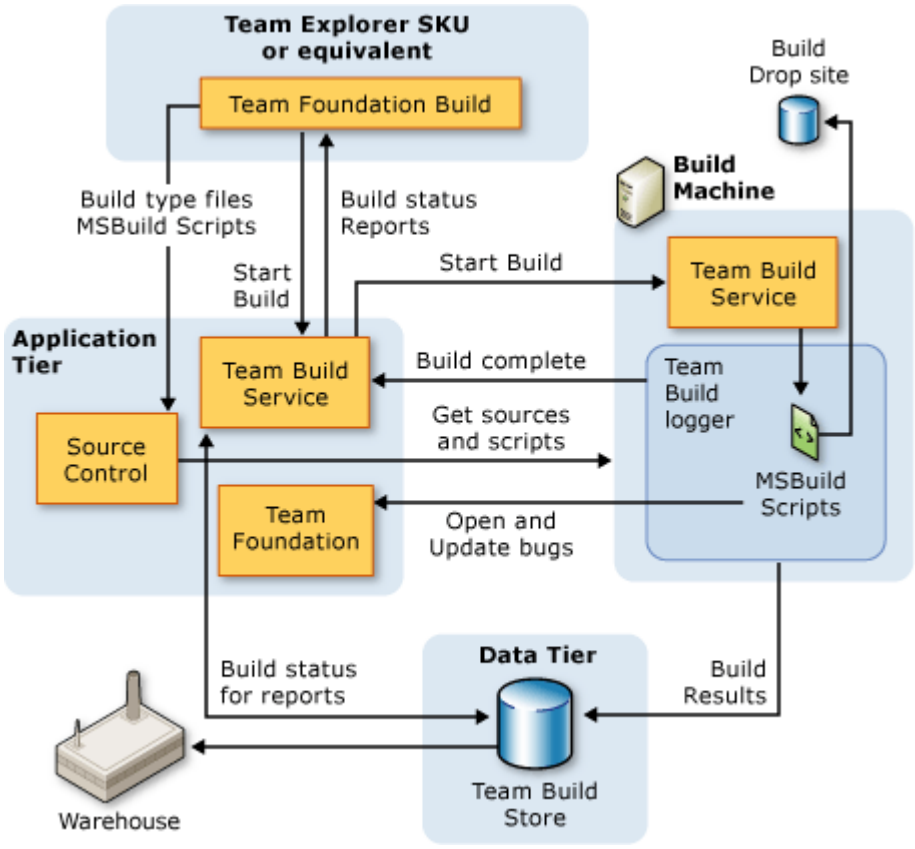
3.2.1 Konkrétní úkoly

Mým úkolem bylo nastudovat možnosti nového nástroje pro testování Microsoft Test Manager. Obzvláště jsem se měl zaměřit na testování klientského kódu webových aplikací a jeho podporou nástrojem jak ze strany testera, tak ze strany vývojáře řešícího nalezenou chybu. Dále bylo mým úkolem nastudování možností propojení nástroje s platformou pro týmovou spolupráci Visual Studio TFS. Měl jsem vyzkoušet možnosti propagování chyb nalezených testerem a informací nasbíraných při běhu testu do TFS. Nakonec jsem měl znalosti a dovednosti získané samostudiem

a experimentováním s novým nástrojem odprezentovat svému konzultantovi odborné praxe. Tyto úkoly mi zabraly asi 6 dní práce.

3.3 Microsoft Visual Studio Team Foundation Build

Team Foundation Build (zkr. Team Build) je součást balíku nástrojů pro řízení životního cyklu softwarových aplikací Microsoft Visual Studio, která je plně provázána s platformou Visual Studio TFS. Team Build je systém pro automatizování sestavení aplikace založený na nástroji Microsoft Build Engine (zkr. MSBuild), což je univerzální systém pro sestavení aplikací všech řízených .NET projektů, dodávaný s rozhraním .NET Framework. Team Build poskytuje výchozí proces sestavení vhodný pro většinu .NET aplikací, který však lze upravit a rozšířit podle specifických požadavků aplikace. Pomocí systému Team Build lze sestavování aplikace spustit manuálně, plánovaně nebo v rámci metody kontinuální integrace, kdy se sestavení provádí při každém vrácení kódu se změnami do správy zdrojového kódu. Architektura systému Team Build je odvozena z architektury platformy TFS, jejíž je součástí. Následuje obrázek zobrazující architekturu systému Team Build a seznam se stručným popisem jeho hlavních částí:[5]



Obr. 3-3: Architektura systému Team Foundation Build[6]

- **Team Build client** – sada Visual Studio poskytuje několik klientů umožňujících práci se systémem Team Build. Jsou jimi zásuvný modul sady Visual Studio Team Explorer, klient příkazového řádku TFSBuild.exe a webové rozhraní TFS Web Access. Team Build také nabízí rozhraní API, pomocí kterého lze vytvořit klienta vlastního.

- **Build Agent** – počítač s nainstalovanou službou Team Build vykonávající proces sestavení aplikace. Každý Build Agent může být asociován nejvýše s jednou aplikační vrstvou TFS.

- **TFS application tier** – aplikační vrstva platformy TFS je implementovaná jako sada Webových služeb hostovaných službou IIS. Každý Team Build klient, který chce komunikovat se systémem Team Build nebo s Build Agentem, tak musí učinit skrz aplikační vrstvu platformy TFS.

- **TFS data tier** – datová vrstva platformy TFS tvořená několika relačními databázemi systému Microsoft SQL Server. Z pohledu systému Team Build obsahuje dvě hlavní databáze, TFSBuild a TFSWarehouse.

- **TFSBuild database** – databáze, která uchovává data o jednotlivých sestaveních, např. seznam Build Agentů, definice sestavení nebo historii sestavení.

- **TFSWarehouse database** – databáze uchovávající historická data o jednotlivých sestaveních pro učely reportingu. Data jsou zde uchována i po jejich odstranění z databáze TFSBuild.

- **Drop folder** – po dokončení sestavení jsou záznamy o sestavení, výstup sestavení a výsledky testů zkopírovány do sdílené síťové složky.

Skripty systému MSBuild a potažmo Team Build jsou soubory ve formátu XML obsahující popis jednotlivých kroků prováděných při sestavení. Jelikož tedy Team Build vychází z MSBuild je pro celkové pochopení nutné nastudovat základy a možnosti systému MSBuild. Systém Team Build je velice komplexní a navržen tak, aby bylo možné změnit nebo rozšířit téměř každou jeho část s ohledem na požadavky vyvíjené aplikace. V tomto oddíle jsme systém Team Build krátce uvedli a popsali jeho nejdůležitější součásti.

3.3.1 Konkrétní úkoly

Mými konkrétními úkoly bylo nastudovat a vyzkoušet automatické sestavení webové aplikace na straně platformy TFS. Dále jsem měl zjistit možnosti automatického nasazení webové aplikace na webový server, zjistit zda je možné provést nasazení SQL skriptů do databáze systému Microsoft SQL Server pod určitým uživatelem, spouštět testovací scénáře a provádět modifikaci konfiguračního

souboru Web.config při nasazení webové aplikace na webový server, to vše v rámci automatického sestavení. Tyto úkoly mi zabraly asi 10 dní práce.

4 Řešení úkolů

4.1 Systém pro automatické testování webových aplikací

Řešení tohoto úkolu jsem začal studiem fungování webových aplikací ASP.NET, o kterých jsem toho před začátkem stáže věděl jen opravdu málo. Také jsem si první semestr zapsal předmět Vývoj internetových aplikací, kde jsem získal další potřebné znalosti. Zároveň jsem se snažil nasbírat co nejvíce vědomostí o funkčním testování webových aplikací, což vyústilo objevením nástrojů společnosti Microsoft týkajících se této oblasti. Další studium záležitostí okolo zadaného úkolu se týkalo téměř výhradně zkoumání a hledání informací a příkladů jejich použití. Nainstaloval jsem proto nejnovější a aktuálně dostupnou sadu Microsoft Visual Studio 2010 Ultimate Beta2 a jako testovací aplikaci jsem použil svůj semestrální projekt ze zmiňovaného předmětu. Mohl jsem začít se zkoumáním a experimentováním s Web Testy. Na své testovací webové aplikaci jsem postupně zkoušel základní možnosti testování pomocí Web Testů. Naučil jsem se jak nahrát, popř. manuálně vytvořit Web Test, seznamoval jsem se uživatelským rozhraním a možnostmi součástí Web Testu a nejvíce jsem věnoval pozornost širokým možnostem modifikace testu pomocí Web Test Editoru. Zkoušel jsem chování jednotlivých předpřipravených pravidel jak typu Validation Rule, tak i typu Extraction Rule. Zkoušel jsem si také naimplementovat testovací zásuvné moduly Web Testu i požadavků Web Testu, zkoušel jsem nakonfigurovat napojení Web Testu na zdroj dat a dále jej v testu používat. Dále jsem zkoušel připravené testy spouštět oproti mé testovací aplikaci a pokoušel se z jejich výsledků získat co nejvíce relevantních informací. Také jsem si vyzkoušel vygenerování Coded Web Testů, kde jsem studoval základy rozhraní API Web Testu. Získané znalosti a dovednosti jsem poté prezentoval svému konzultantovi, který mi následně zadal další úkoly k nastudování a vyzkoušení.

Dále bylo mým úkolem vyzkoušet Web Testy na reálné webové aplikaci distributora knižních titulů a zjistit, zda lze porovnat celkový počet kusů veškerého sortimentu zobrazovaného webovou aplikaci s počtem kusů vráceným uloženou procedurou z databáze. Začal jsem tvorbou jednoduchých Web Testů a následně jsem je oproti reálné webové aplikaci spouštěl a sledoval výsledky. Poté jsem se pustil do druhé části úkolu, který jsem úspěšně vyřešil pomocí vlastního pravidla typu Validation Rule. Podle příkladu z dokumentace jsem vytvořil pravidlo vlastní a začal jsem s implementací požadované funkcionality. Z HTML kódu stránky zobrazující veškerý sortiment jsem potřeboval získat hodnotu určující celkový počet kusů, což se mi podařilo s využitím jazyka Xpath a regulárních výrazů. Pomocí jazyka Xpath jsem byl schopen vyextrahovat text obsahující hodnotu z těla stránky HTTP odpovědi. Použití regulárních výrazů bylo nutné, protože řetězec vyextrahovaný ze stránky se skládal z textu a taky několika čísel, z nichž právě jedno určovalo požadovaný celkový počet

záznamů. Uloženou proceduru databáze jsem zavolal běžným postupem pomocí rozhraní ADO.NET. Těmto parametrům bylo ještě nutné vytvořit veřejné vlastnosti, čímž je lze zpřístupnit v uživatelském rozhraní Web Test Editoru. Následující výpis kódu zobrazuje vytvořené vlastní pravidlo typu Validation Rule. Výpis neobsahuje atributy zobrazující popis pravidla a jeho vlastností v UI Web Test Editoru.

```
public class CompareValueWithSprocValidationRule : ValidationRule
{
        public string Xpath { get; set; }
        public string RegexFilter { get; set; }
        public string ProcedureName { get; set; }

        public override void Validate(object sender, ValidationEventArgs e)
        {
            try
            {
                HtmlAgilityPack.HtmlDocument htmlDoc =
                    new HtmlAgilityPack.HtmlDocument();
                htmlDoc.LoadHtml(e.Response.BodyString);
                HtmlAgilityPack.HtmlNode node =
                    htmlDoc.DocumentNode.SelectSingleNode(Xpath);

                if (node == null) throw new ApplicationException
                    („Xpath Query: Specified tag not found in 15ese response.“);

                Regex 15es = new Regex(@RegexFilter);
                Match match = 15es.Match(node.InnerHtml);

                using (SqlConnection connection = new SqlConnection())
                {
                    connection.ConnectionString =
                        Properties.Settings.Default.MyPemicConnectionString;
                    connection.Open();

                    using (SqlCommand command =
                        new SqlCommand(ProcedureName, connection))
                    {
                        command.CommandType = System.Data.CommandType.StoredProcedure;
                        string result =
                            Convert.ToString(command.ExecuteScalar());

                        if (match.Value == result)
                            e.IsValid = true;
                        else
                            e.IsValid = false;
                    }
                }
            }
            catch (Exception ex)
            {
                e.IsValid = false;
                e.Message = ex.Message;
            }
        }
    }
```

Pro převedení těla stránky do objektové reprezentace DOM jsem použil parser HtmlAgilityPack, který si poradí i s běžně malformovaným HTML kódem. HtmlAgilityPack je knihovna tříd vydaná pod licencí Ms-PL doporučovaná samotnými odborníky ze společnosti Microsoft. Popsaný úkol jsem vyřešil pomocí vlastního validačního pravidla, čímž jsem se také naučil využívat základní možnosti rozšíření Web Testu. Tuto dovednost jsem následně využil při řešení dalších úkolů.

Po odprezentování řešení svému konzultantovi bylo mým úkolem předvést testerovi možnosti Web Testů na výše zmiňované webové aplikaci. Tester mi poté sdělil své dojmy ze systému a požadavky, které by mu mohly ušetřit velké množství času, jinak stráveného zdlouhavým manuálním testováním. Nakonec mi představil některé konkrétní testovací scénáře a domluvili jsme se, že mi připraví několik testovacích scénářů, které se pokusím naimplementovat, čímž se hlouběji ověří možnost použití Web Testů na reálné webové aplikaci. Jakmile mi tester připravil testovací scénáře, začal jsem s prací.

Jednodušší testovací scénáře se týkaly ověření správné funkce filtru sortimentu, kde jsem měl zvolit některé jeho volby a po potvrzení ověřit správnost zobrazeného výsledku. U všech položek na všech stránkách výpisu sortimentu bylo nutné zkontrolovat, zda hodnoty v určitém sloupci tabulky odpovídají výběru filtru. Například jsem měl vybrat pouze knižní tituly v anglickém jazyce a zkontrolovat, zda se ve výsledku zobrazí pouze odpovídající položky s ikonou anglické vlajky a popiskem Angličtina ve sloupci s názvem Jazyk. Posledním krokem testovacího scénáře bylo ověřit celkový počet zobrazených záznamů s celkovým počtem vráceným uloženou procedurou. Při řešení jsem byl nucen napsat si vlastní pravidlo typu Conditional Rule reprezentující klasický cyklus for, známý například z jazyka C#. Odpovídající pravidlo je již součástí rámce Web Test Framework, avšak pro některé jeho vlastnosti lze použít pouze číselné hodnoty a ne parametry kontextu Web Testu. To nepřicházelo v úvahu, protože jsem cyklus potřeboval použít k procházení všech stránek výpisu sortimentu, jejichž počet se mění podle dané situace. Vestavěný cyklus umožňuje pouze statické zadání počtu iterací při modifikaci testu ve Web Test Editoru. Já jsem naopak potřeboval počet iterací získat z těla stránky a dosadit jako vlastnost cyklu za běhu testu, na což se právě velice dobře hodí parametry kontextu Web Testu. Po úspěšném naimplementování vlastního cyklu bylo nutné vytvořit jedno pravidlo typu Extraction Rule a jedno pravidlo typu Validation Rule. Pravidlem typu Extraction Rule jsem musel ze stránky získat počet stránek výpisu sortimentu a následně jej dosadit jako ukončovací hodnotu cyklu. Pomocí pravidla typu Validation Rule jsem na jednotlivých stránkách výpisu kontroloval přítomnost prvků vyžadovaných testovacím scénářem, tj. v tomto případě zobrazení správných ikonek a jejich popisků. Sortiment je na stránkách zobrazen pomocí HTML tabulky a tímto pravidlem jsem docílil validace obsahu daného sloupce u každého řádku tabulky. K tomu mi opět dopomohl jazyk Xpath a regulární výrazy. Tentokrát jsem pomocí Xpath dotazu potřeboval vrátit kolekci všech odpovídajících elementů HTML dokumentu a ne pouze jeden konkrétní element, jako tomu bylo u předchozího validačního pravidla. Tím jsem z tabulky vybral obsah všech buněk daného sloupce. Buňky obsahovaly pouze HTML element img pro obrázek, u kterého jsem porovnal hodnotu atributu src s URL adresou obrázku zadanou jako vlastnost pravidla. Aby test dopadl úspěšně, musí se hodnoty zmiňovaného atributu u všech řádků tabulky na všech stránkách výpisu shodovat se zadanou hodnotou. Pomocí několika vlastních pravidel různých typů se mi tedy podařilo plně automatizovat vykonávání prvního testovacího scénáře.

Poté jsem implementoval další testovací scénáře ze sady od testera. U některých bylo možné použít některá již hotová vlastní pravidla, u jiných jsem musel vytvořit další vlastní pravidla podle jednotlivých požadavků. Zatím jsem vytvořil knihovnu tříd s 8 pravidly a některé z nich postupně upravuji, aby se staly univerzálnějšími a ne pouze specifickými pro daný testovací scénář. Ve výsledku by měla vzniknout knihovna znovupoužitelných pravidel, která by se s dalšími specifickými požadavky rozšiřovala. Celkem jsem naimplementoval 10 testovacích scénářů, což může nahradit jejich zdlouhavé ruční provádění. Hotové testy lze plánovaně automaticky spouštět, např. v noci. Tester si poté po příchodu do zaměstnání zobrazí výsledky již hotových testů a po zbytek pracovní doby se může věnovat jiným činnostem. Další výhodou je, že automatické vykonávání testovacích scénářů je preciznější než manuální. Například stránek výpisu sortimentu ke kontrole může být několik stovek a při manuálním testování tester prochází a kontroluje jen několik prvních z nich, kdežto automatizovaný test projde úplně všechny stránky i všechny jejich položky. Při implementaci testovacích scénářů je možné využít i Coded Web Testy bez vytváření vlastních pravidel a veškerou logiku psát přímo pomocí kódu. Protože většina testerů nejsou programátoři, snažil jsem se, aby k vytváření testů nebylo zapotřebí znalosti programování a co nejvíce tak využil uživatelského rozhraní Web Test Editoru. To se podařilo, jelikož všechny vytvořené testy jsou „naklikané“ bez napsání jediného řádku kódu. Testerovi tak zbývá pouze seznámit se s funkcí a prostředím Web Testů a naučit se základy dotazovacího jazyka Xpath a regulárních výrazů, bez čehož se při práci s Web Testy neobejde.

4.2 Microsoft Test Manager

Mým úkolem bylo prozkoumat možnosti nového nástroje Microsoft Test Manager, což jsem provedl. Tento nástroj je provázán s platformou TFS, kterou jsem musel nainstalovat na svůj počítač a naučit se používat její základní funkce. Naučil jsem se ovládat správce Source Control, Team Explorer, vyzkoušel jsem si webové rozhraní TFS Web Access, vytvářel jsem definice sestavení, zadával bugy a podobně.

Poté jsem nainstaloval nástroj Microsoft Test Manager a začal s jeho studiem a konfigurací. Zprovoznil jsem testovací řadič s jedním testovacím agentem a vytvořil jsem si minimální fyzické prostředí o jednom počítači, abych mohl testy vůbec spustit. Testovací scénáře jsem spouštěl asociované s manuálními i automatizovanými testy. Zkoumal jsem data a informace nasbírané diagnostickými datovými adaptéry za běhu testu. Nakonfigurovaný nástroj, spouštění testů a jejich výsledky jsem předvedl svému konzultantovi.

Dále jsem měl zjistit, jak je na tom tento nástroj s podporou testování klientského kódu JavaScript webových aplikací. Nástroj Microsoft Test Manager umožňuje správu a spouštění automatizovaných testů typu Web Test, avšak Web Testy neumožňují testovaní klientského kódu. Musel jsem vyzkoušet jinou možnost a tak jsem se zaměřil na manuální testování webové aplikace. Nový nástroj sice poskytuje efektivní prostředí pro vykonávání manuálních testů, ale jak jsem z dostupných zdrojů výrobce zjistil, JavaScript není v aktuální verzi nástroje Microsoft Test Manager podporovanou technologií.

4.3 Microsoft Visual Studio Team Foundation Build

Při řešení tohoto úkolu jsem využil základních znalostí platformy TFS získané prací na předchozím úkolu. Tentokrát jsem se ale mnohem více zaměřil na možnosti automatického sestavení webových aplikací pomocí systému Team Build. Protože tento systém vychází se systému MSBuild, začal jsem nejprve studiem jeho možností, což je pro zvládnutí systému Team Build nezbytné. Poté jsem začal studovat architekturu systému Team Build a jeho výchozí proces sestavení, který je vhodný pro většinu běžných aplikací. Mně zadané úkoly však vyžadovaly rozšíření tohoto výchozího procesu o některé specifické požadavky. Ve své testovací solution jsem vytvořil konfigurační soubor automatického sestavení TFSBuild.proj. Pomocí tohoto souboru jsem zkoušel funkci jednotlivých vestavěných tasků, což jsou příkazy pro systém Team Build při vykonávání sestavení aplikace. V konfiguračním souboru sestavení jsem použil task exec, který spouští program s parametry. Požadovaný program lze definovat pomocí atributu Command. V rámci sestavení jsem potřeboval vykonat SQL skript na databázi pod určitým uživatelem a k tomu jsem použil utilitu sqlcml, která umožňuje pracovat se systémem Microsoft SQL Server. Pro nasazení výstupu sestavení na webový server jsem použil utilitu xcopy. Obsah cílové složky pro nasazení jsem nejprve pomocí tasku CleanFolder vymazal a nakopíroval potřebné soubory. Protože soubor Web.config je ve formátu XML, použil jsem pro jeho úpravu před nasazením aplikace XSLT transformaci pomocí tasku XslTransform, který je součástí volně dostupné knihovny tasků Microsoft.Sdc.Tasks.dll. Nakonec jsem do testovacího kontejneru sestavení přidal testy, které se mají vykonat oproti právě nasazené testovací aplikaci.

5 Závěr

5.1 Získané znalosti

Během odborné praxe ve firmě jsem získal velké množství znalostí a zkušeností. Materiály k nastudování jsem dostal před prvním dnem nástupu a do jejich čtení jsem pustil už při zpáteční cestě z přijímací schůzky. O webových aplikacích ASP.NET jsem toho věděl opravdu málo a o jejich testování ještě míň. Ovládal jsem základní znalost jazyka C# a tvorby konzolových i Windows Forms aplikací díky absolvovanému předmětu na toto téma. Ze začátku jsem byl v záplavě nových informací ztracen, což se zlepšovalo jejich postupným vstřebáváním a hlavně díky práci na úkolech týkajících se reálné aplikace. Nejvíce jsem určitě získal ve fázi implementace testovacích scénářů, kdy jsem dostal konkrétní zadání a požadavky, které mám zpracovat. Web Testy mi umožnily pohlédnout na webové aplikace jako na jakési rozhraní, které je pro uživatele běžně skryto za uživatelským rozhraním zobrazeným webovým prohlížečem. Web Testy tedy nejsou nic jiného než skripty nějak využívající tohoto rozhraní. Z tohoto pohledu jsem se během praxe o webových aplikacích dozvěděl podstatné množství zásadních informací. Díky tomu mi záležitosti týkající se fungování webových aplikací dnes připadají zcela zřejmé. Dále jsem získal poměrně dostatek vědomostí o základních i některých pokročilých funkcích a vlastnostech nástrojů pro řízení životního cyklu softwarových aplikací od společnosti Microsoft. To vše mi jako začínajícímu programátorovi na platformě .NET podstatně rozšířilo rozhled a poskytlo solidní základ pro budoucí praxi a sebezdokonalování v těchto i dalších oblastech.

5.2 Chybějící znalosti

Na začátku odborné praxe jsem postrádal téměř veškeré nutné znalosti kromě základní znalosti programování v jazyce C#. Chyběly mi znalosti o fungování webových aplikací, o testování aplikací a použití týmových nástrojů. S pochopením webových aplikací mi v prvním semestru výkonu praxe pomohl absolvovaný předmět Vývoj internetových aplikací, kde jsem získal potřebný základ. Podklady a materiály ke všem ostatním zadaným úkolům jsem si musel najít a nastudovat už samozřejmě sám.

5.3 Výsledky a celkové zhodnocení

Všechny zadané úkoly jsem splnil a výsledky představil svému konzultantovi Radku Garzinovi. Prozatím proběhly dvě konzultace mé práce na testovacích scénářích s testerem. V nejbližších dnech ještě budu mít prezentaci na téma Web Testing pro odborníky z webového oddělení. Celkově odbornou praxi hodnotím pozitivně a pro můj budoucí profesionální život ji považuji za velký přínos a hodnotnou zkušenost.

Literatura

1. *KVADOS, a.s.* [online]
   URL: <http://www.kvados.cz/Ospolečnosti/Onás/tabid/53/Default.aspx> [citováno 24. dubna 2010]

2. *Ed Glas's blog on VS load testing: Web Test Authoring and Debugging Techniques for VS 2010* [online]
   URL: <http://blogs.msdn.com/edglas/archive/2010/03/24/web-test-authoring-and-debugging-techniques-for-visual-studio-2010.aspx> [citováno 24. dubna 2010]

3. *VS 2010 Application Lifecycle Management: Binding a Data Source to a Web Performance Test* [online]
   URL: <http://msdn.microsoft.com/en-us/library/ms404707.aspx> [citováno 25. dubna 2010]

4. *VS 2010 Application Lifecycle Management: Defining Your Testing Effort Using Test Plans* [online]
   URL: <http://msdn.microsoft.com/en-us/library/dd286581.aspx> [citováno 26. dubna 2010]

5. HASHIMI, Sayed Ibrahim; BARTHOLOMEW, William. *Inside the Microsoft Build Engine: Using MSBuild and Team Foundation Build*. Redmond (Washington) : Microsoft Press, 2009. 406 s.

6. *Visual Studio Team System: Team Foundation Build Overview* [online]
   URL: <http://msdn.microsoft.com/en-us/library/ms181710(VS.90).aspx> [citováno 26. dubna 2010]

7. *MSDN Library* [online]
   URL: <http://msdn.microsoft.com/cs-cz/library/default.aspx>

8. *Ed Glas's blog on VS load testing: Content Index for Visual Studio Web Tests and Load Tests* [online]
   URL: <http://blogs.msdn.com/edglas/pages/content-index-for-web-tests-and-load-tests.aspx>